AN OVERVIEW ON BEHAVIOUR-BASED METHODS FOR AUV CONTROL

M. Carreras[ψ], J. Batlle[ψ], P. Ridao[ψ] and G.N. Roberts[#]

ψ Computer Vision and Robotics Group, University of Girona
Edifici Politècnica II, Campus Montilivi, 17071 Girona, Spain
{marcc, jbatlle, pere}@eia.udg.es

Mechatronics Research Centre, University of Wales College, Newport
Allt-yr-yn Campus, PO Box 180, Newport, South Wales, NP20 5XR, UK
g.roberts@newport.ac.uk

Abstract: Traditionally, deliberative reasoning methods based on a world model have been used to guide Autonomous Vehicles. Nevertheless, due to serious problems when dealing with complex and changing environments the alternative approach of reactive architectures appeared, directly coupling perception and action without any world representation. This paper describes the characteristics and variants of reactive architectures as well as four of the most suitable ones to guide Autonomous Underwater Vehicles: Schema-based approach, Subsumption, Process Description Language and Action Selection Dynamics. Each architecture is described and evaluated using an underwater simulator with an AUV model.

Keywords: Autonomous Vehicles, Artificial Intelligence, Control Systems, Marine Systems, Robot navigation, Vehicle Simulation.

1. INTRODUCTION

A great deal of work concerning Autonomous Underwater Vehicles has been carried out in recent years, mainly when human supervision is not strictly necessary. In this context, one of the most important subjects for developing the autonomy of the vehicle is the design of the control architecture. There are three different kinds of architectures. The first is deliberative architectures, which are based on planning using a world model. A mission is specified to achieve a set of goals, and each goal is executed by a control system. These architectures are suitable for structured and highly predictable environments. Nevertheless in complex, non-structured and changing environments, like the majority of underwater environments, serious problems appeare while trying to maintain, in real-time, an accurate world model. The second, the reactive architectures, appeared in response to the difficulty of the deliberative architectures to deal with non-structured and dynamic environments. As Arkin said, "Reactive control is a technique for tightly coupling perception and action, typically in the context of motor behaviours, to produce timely robotic response in dynamic and unstructured worlds" (Arkin, 1998). This technique tries to avoid any kind of representation of the world. Basically, a group of parallel behaviours act independently generating outputs to the actuators according to the value of the sensors. For this reason they are also called Behavioural architectures. Reactivity, real-time response and easy implementation are the most relevant advantages. The disadvantages are the difficulty in designing a set of behaviours in order to plan and to specify high-level goals. Finally, there are hybrid architectures which take advantage of both deliberative and behavioural approaches. Usually, they are structured in three layers: the deliberative layer based on planning, the control execution layer and the functional reactive layer. Hybrid architectures are the most frequently used in AUV's and are also found in autonomous air and land vehicles. Many comparative studies can be found concerning control architectures for AUV's: (Coste-Maniere, *et al.*, 1995), (Valavanis, *et al.*, 1997) or more recently (Ridao, *et al.*, 1999).

This paper is structured as follows. Section 2 reviews the most important characteristics of behaviour-based architectures. Section 3 describes the simulation environment and the underwater vehicle model used. Section 4 explains four important behavioural architectures. Methodology and formalisms to implement them are shown with examples and simulation results. Section 5 summarises the work presented with the conclusions of the paper.

2. BEHAVIOUR-BASED ROBOTICS.

A reactive robotic system tightly couples perception to action without the use of intervening abstract representations or time history. The main features of a reactive robotic system are:

- The behaviours are basic building blocks to carry out robotic actions.
- The generation of responses is without representation of the knowledge.
- Systems are inherently modular allowing the addition of new behaviours without redesigning.

As reactive architectures are built with behaviours they are also called Behaviour-based architectures. There are different ways to define a behaviour system. Hereafter, it will be shown how to express, design, encode and assemble a behaviour. This will enable understanding of how to design a behaviour-based architecture or control layer. Further readings can be found in (Arkin, 1998) and (Pfeifer, *et al*., 1999).

Behaviours can be expressed as a Stimulus-Response block. The global architecture is represented with behaviours in parallel structure and outputs are channelled into a coordinate mechanism that produces an appropriate response (Fig. 1). Also, behaviours can be expressed with a functional notation, like mathematical functions. A Coordination function evaluates all the behaviours and gives the final response.

2.1. Behavioural choice and design

There are many approaches to choose and design behaviours. Three of the most known are described next.

2.1.1. Ethologically guided/constrained design

This method is based on animal behaviour. Behaviours are designed by consulting biological literature and searching animal behaviours, which can be used for a robotic system. An animal model, which accomplishes the wanted behaviour, is translated to a more suitable model to be implemented in the robot. The schema-based robotic navigational system was invented based on the navigational behaviour of the toad.

2.1.2. Situated activity-based design

Robot actions are predicted upon the situations in which the robot is located. The design requires a solid understanding of the relationship between the vehicle and its environment. Then all possible situations must be related with behaviours. The problem is reduced to recognising the situation of the robot.

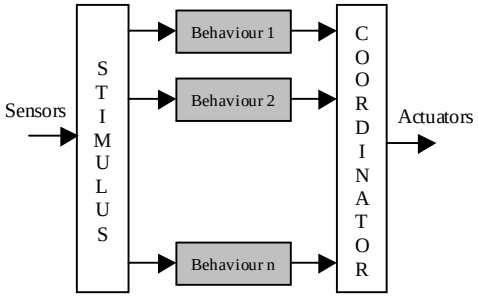

Fig. 1. Behaviour based architecture.

2.1.3. Experimentally driven design

Bottom-up approach. The basic premise is to build up a set of competences and add them to the behaviour system after a test process. By repeating this process iteratively, the behaviour-based architecture is built.

2.2. Behavioural encoding

To encode the behavioural response it is necessary create a functional mapping from the *stimulus plane* to the *motor plane*. The motor plane usually needs two parameters, the strength (magnitude of the response) and the orientation (direction of the action). A behaviour can be expressed as (S, R, β) where:

- *S: Stimulus Domain. S* is the domain of all perceivable stimuli. A stimuli is represented by $s(p,\lambda)$, where p is the class of perception and λ its strength. Each class p has a τ threshold value above which a response is generated.
- *R: Range of Responses*. For autonomous vehicles with six degrees of freedom, the response $r \in R$ of a behaviour is a six-dimensional vector: r = [x, y, z, θ, ϕ, ψ] composed by the three translation degrees of freedom and the three rotational degrees of freedom. Each parameter has strength and orientation. The response of each behaviour is modified depending on its importance in respect to the others.
- *β: Behavioural Mapping.* For each behaviour the mapping function β relates the stimulus domain with the response range: $\beta(s) \rightarrow r$. The mapping function generates response only when $\lambda > \tau$. Behavioural mappings β can be :
  - *Null*: the stimulus produces no motor response
  - *Discrete*: numerable set of responses; i.e. Subsumption architecture.
  - *Continuous*: infinite space of potential reactions; i.e. potential fields.

2.3. Assembling behaviours

When combining and coordinating multiple behaviours the emergent behaviour appears. This is the product of the complexity of the relationship between a robotic system and the real world. The coordination function assembles all the active behaviour responses and generates a final vector response. Final output can be an assemblage of behaviour aggregations or other assemblages. The two primary coordination mechanisms are:

- Competitive methods: the output is the selection of a single behaviour. Principal methods are *suppression network* (subsumption architecture), *action-selection* and *voting-based coordination*.
- Cooperative methods: the output is the superposition of force gradients. Principal methods are *vector summation* (potential fields), *constrain of control variables* and *behavioural blending* (fuzzy variation).

3. SIMULATION ENVIRONMENT

To evaluate the control architectures, a simulated AUV model and virtual environment was implemented using Matlab/Simulink. An AUV can be modelled by two approaches (Goheen, 1995) predictive techniques and testing techniques. The model used in these simulations is based on predictive techniques which build up the model from the main physical laws governing the dynamic behaviour of the system. The structure of the dynamic model is shown in equation 1, (Fossen, 1995).

$$A = (M_{RB} + M_A)^{-1}(T + G - D(V)V - (C_{RB}(V) + C_A(V))V) \quad (1)$$

where:

A: accelerations vector
V: speeds vector
M_{RB}: inertia matrix of the rigid body
M_A: inertia matrix of the added mass
T: input forces and torques vector
G: gravity and buoyancy vector
D(V): Hydrodynamic damping matrix
$C_{RB}(V)$: coriolis matrix of the rigid body
$C_A(V)$: coriolis matrix of the added mass

The input of the dynamic equation is a vector with a force and a torque for each axis of the onboard coordinate system. These forces and torques are obtained by addition of the individual forces and torques exerted by the thrusters. Once the acceleration of the AUV is calculated this is integrated to obtain speed. The model used for the simulations was developed for the underwater vehicle called GARBI, figure 2. GARBI (Amat, *et al.*, 1996) was first conceived as a low-cost Remotely Operated Vehicle (ROV) for exploration in waters up to 200 meters in depth. At present work is in progress on the implementation of a new control architecture named OOCAA (Object Oriented Control Architecture for Autonomy). The main goal is to use this architecture to transform the GARBI into an AUV.

In order to visualise the 3-dimensional behaviour of the vehicle, a virtual environment has been implemented. Underwater surfaces are introduced to evaluate the response of the control architectures. The vehicle is simulated with seven sonars that perceive the vicinity objects. Figure 3 shows an image of the underwater environment.

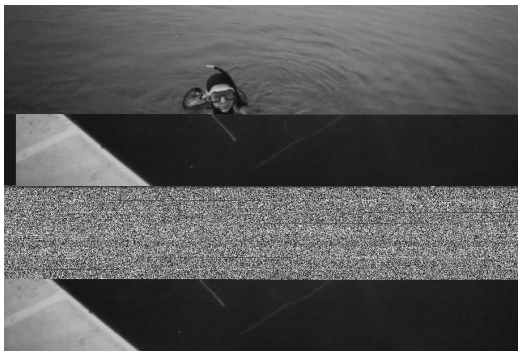

Fig. 2. GARBI underwater vehicle.

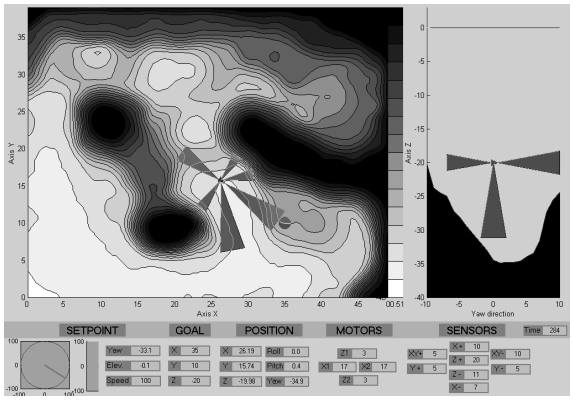

Fig. 3. Simulated underwater environment

4. BEHAVIOUR-BASED METHODS FOR AUV'S

Many behaviour-based architectures can be applied to an AUV. Examples of typical behaviours are: go to, obstacle avoidance, wander, station-keeping, target-following, target-tracking, avoid trapping and bottom-following. In this section four of the most conceptually important architectures are described and evaluated with simulations. In each architecture 3 behaviours are implemented: *go to, obstacle avoidance* and *avoid trapping*. The behaviours generate a 3-dimensional vector (defined by two angles and a magnitude) that denotes the direction to be followed by the vehicle. The *go to* behaviour (Fig. 4) generates a vector with a constant magnitude pointing at the point goal. The *obstacle avoidance* behaviour (Fig. 5) generates a vector with a variable magnitude depending on proximity of the obstacles detected by sonars. The direction of the vector is opposite to the objects. The *avoid trapping* (Balch, 1993) behaviour (Fig. 6) saves a map of the recent path of the vehicle. If the vehicle remains in the same zone for a long period the behaviour becomes active and generates a vector in the opposite direction of the recent path with a magnitude proportional to the visited zone. The reactive architecture then selects the final behaviour to be followed. A controller system is in charge of driving the vehicle in such direction.

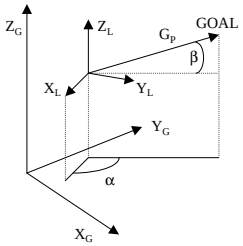

Fig. 4. "Go To" behaviour.

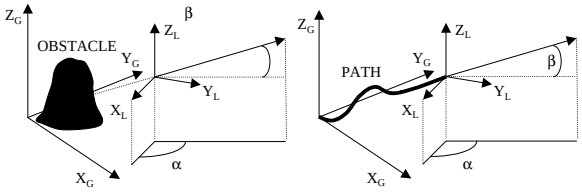

Fig. 5 and 6. "Obstacle avoidance" and "Avoid trapping" behaviours respectively.

4.1. Schema-based approach

4.1.1. Description

This architecture is the most intuitive of the four examined. *Motor schema* is the basic unit from which complex actions are constructed. Each schema operates as a concurrent, asynchronous process initiating a behavioural intention. Motor schemas react proportionally to sensory information perceived from the environment. In order to give more priority to some schemas the output vector is multiplied by a gain value. Safety or dominant behaviours must have higher gain values. The coordination method consists of vector summation of all motor schema outputs and normalisation. The set of all coordination outputs for all positions inside the environment constitute a potential field. General reference (Arkin, 1989), 3-dimensional reference (Arkin, 1992), AUV reference (Warren, 1990).

4.1.2. Implementation

Implementation is achieved by defining a set of behaviours with a motor schema library. Schemas have internal parameters depending on the behaviour and an external parameter, the gain value that gives priority to the behaviours. Each schema can be executed into a different processor. Nevertheless the outputs must have the same format in order to be summed by the coordinator. Techniques like fuzzy logic can be used in the implementation of the motor schemas or in the coordination phase.

4.1.3. Results

The motor schema architecture has been implemented and tuned with the 3 behaviours (Figure 7). Obstacle avoidance behaviour has the higher gain value followed by avoid trapping and goto. Principal advantages are simplicity and easy implementation as well as optimised paths (Figure 8). Difficulties appear in tuning the gain values. When new behaviours are added, re-tuning is necessary. This architecture has been tested in reality (Fig. 2) with GARBI achieving very good results when the gains were tuned. The vehicle reached a goal position avoiding a static obstacle

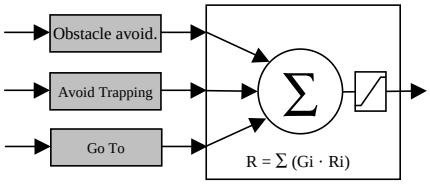

Fig. 7. Schema-based approach architecture.

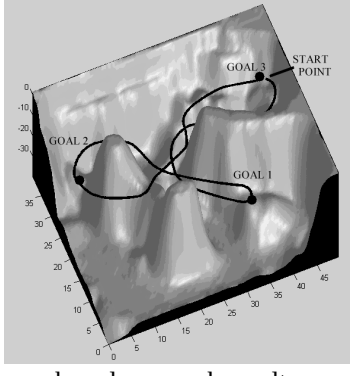

Fig. 8. Schema-based approach results.

4.2. Subsumption architecture

4.2.1. Description

Subsumption is a method of reducing a robot's control architecture into a set of task-achievement behaviours or competences represented as separate layers. Individual layers work on individual goals concurrently and asynchronously. Layers are organised hierarchically allowing higher layers to inhibit inputs or suppress outputs of lower layers. This constitutes the coordination method. The architecture can be built incrementally adding layers in different phases. Each layer is composed of one or more Augmented Finite State Machines (AFSM), and depending on sensory information the layer can be active or not. When a layer is active, its output suppresses all outputs from the layers below taking the control of the vehicle. The layer can remain active for a period after the activation conditions finishes. General references (Brooks, 1986) and (Connell, 1989), AUV reference (Boswell, *et al.*, 1994).

4.2.2. Implementation

Each layer is has one or more AFSM that is composed by a finite state machine with some registers and a set of timers. Registers contain input and output messages of the AFSM. An input message or the expiration of a timer can change the state of the machine. Each behaviour layer can be mapped onto its own processor.

4.2.3. Results

The three behaviours have been implemented using the subsumption architecture (Fig. 9). Each behaviour has been implemented in an AFSM with suppression nodes. In order to assure the safety of the vehicle the *obstacle avoidance* behaviour is the higher layer. Next layer is the *avoid trapping* to take control over the *go to* when vehicle becomes trapped. The principal advantages are robustness, modularity and easy tuning of the behaviours. The disadvantages are the difficulties on the design and the non-optimal trajectories due to the competitive coordination method. Simulation results (Fig. 10).

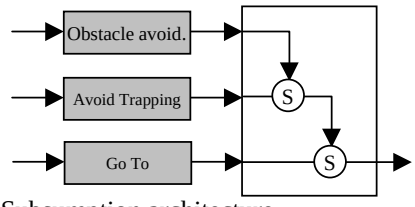

Fig. 9. Subsumption architecture.

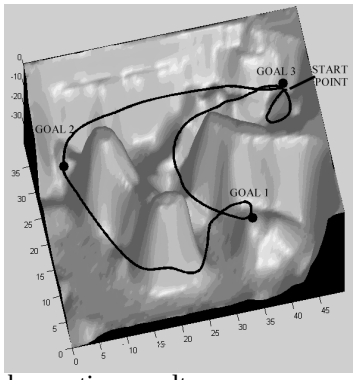

Fig. 10. Subsumption results.

4.3. Process Description Language

4.3.1. Description

PDL is a cooperative dynamics architecture where many active processes operate in parallel. Processes represent behaviours taking information from sensors to generate a control action if ineeded. No control mechanism gives precedence over processes. Instead, each process has a certain influence on some variables, typically the motor speeds. At each time, fixed quantities, of each behaviour, are added or subtracted to the previous output value. All behaviours are always operational. The merged output taken depends on which behaviour process influences more over the others. Emergent behaviours appear by the effect of this merging. PDL works by manipulating derivatives of the variables, this implies that a very fast control loop must be used to prevent that the system becomes unstable. General reference (Steels 1992).

4.3.2. Implementation

PDL can be implemented using a library of process functions. Processes compute sensor inputs and decide if some action must be taken. All the processes must be synchronised and must add his output to the previous merged output. Each process has a threshold for the sensory inputs below which no activation is produced. In order that safety or dominant behaviours take the most part of the control, the fixed value to be added by each process must be tuned. Also the sample time must be small enough to assure that dynamics of the control architecture is faster that dynamics of the vehicle.

4.3.3. Results

PDL control architecture has been implemented (Figure 11). The sample time had to be reduced, compared with that used with three architectures, to faster the dynamics. *Obstacle avoidance* behaviour has the higher addition value followed *by avoid trappin*g and *go to*. Principal advantages are simplicity as well as optimised paths (Figure 12). Disadvantages are the difficulty of tuning and the high sampling frequency.

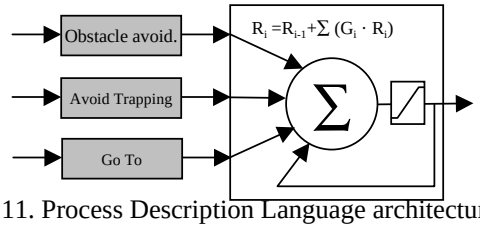

Fig. 11. Process Description Language architecture.

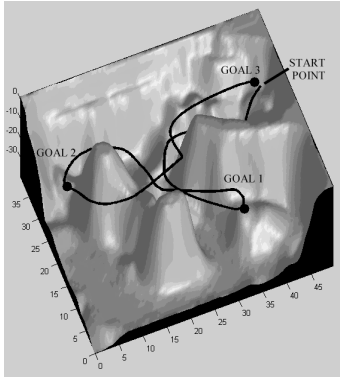

Fig. 12. Process Description Language results.

4.4. Action Selection Dynamics

4.4.1. Description

Action-selection dynamics, is the most complex and uses a dynamic mechanism for behaviour selection. Coordination is achieved by competition. There is a set of competence modules that represent behaviours. The modules are defined by a “condition list”, an “add list” and a “delete list”. When the “condition list” is satisfied the modules can be executed. “Add list” and “delete list” are the effects expected to be true or false respectively. There is a successor link from one module to another if the first has the add list condition that appears also in the condition list of the other. Successor links taken in the other sense are considered as predecessor links. Finally, there is a conflictor link from one module to another if the first has in the condition list a condition that appears also in the delete list of the other. Modules have a level of energy that is modified by different sources. Firstly, the sensor signals provide energy to the modules depending on the environment perceived. If the module has in its add list one of the goals of the architecture, more energy is transferred. Finally, the energy of the modules is spread positively along predecessor and successors, and negatively along conflictors. The module that will be executed is the one that accomplishes the condition list, has the maximum activation energy and this is above a threshold. General reference (Maes, 1990)

4.4.2. Implementation

The coordination module is implemented using a mathematical notation, found in (Maes, 1989), that defines the competence modules characteristics and calculates energy activation. Competence Modules must compute the sensory information and generate the control output that will be coordinated.

4.4.3. Results

Action Selection Dynamics has been implemented and tested (Fig. 13). Figure 14 shows them with the set of links. Robustness and automatic coordination are the principal advantages. Complexity, tuning and difficulty design are the most relevant disadvantages.

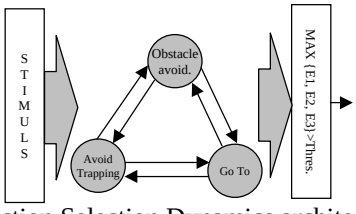

Fig. 13. Action Selection Dynamics architecture.

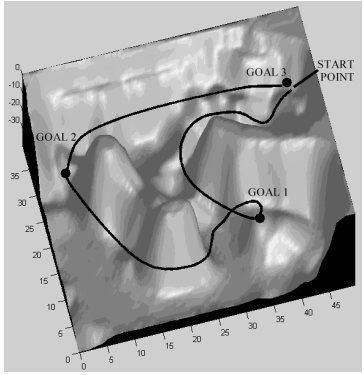

Fig. 14. Action Selection Dynamics results.

4.5. Discussion

After comparing the four evaluated architectures it can be said that each has its own advantages and disadvantages and are well suited for different applications. These advantages can be in different terms. Table 1 summarises some properties with the better architectures to accomplish them:

Property\Architec	**1st**	**2nd**	**3rd**	**4th**
Performance	SCHE.	P.D.L	A.S.D.	SUBS.
Modularity	SUBS	A.S.D.	P.D.L.	SCHE.
Robustness	SUBS	A.S.D.	P.D.L.	SCHE.
Development time	SCHE.	P.D.L.	SUBS.	A.S.D.
Tuning time	SUBS.	A.S.D.	P.D.L.	SCHE.
Simplicity	SCHE.	P.D.L.	SUBS.	A.S.D.

Table 1

Basically, the characteristics of the architectures are related with the coordination method. Competitive methods (Subsumption and A.S.D.) have only one active behaviour. Robustness in dangerous situations is better because only a safety behaviour acts and the danger is eliminated. Also competitive methods have a good modularity and tuning time because behaviours are tuned alone and then mixed. Disadvantages are found in the coordinator design. In order to choose only one active behaviour, a complex method must be used to implement it (A.S.L.) or to understand the hierarchy of behaviours (Subsumption). Finally, bad performance is due to the non-instant merging of behaviours, a lot of bends appear in the path when more than one behaviour is acting consecutively. Nevertheless, cooperative methods have a simple coordinator. Because all behaviours are active they have a good performance achieving optimized paths. Problems appear in the tuning phase. Parameters are critical. In extreme situations non-safe behaviours can annul the safe ones. They have a low modularity because each new behaviour causes re-tuning. Every approach has its own attractiveness that designers must perceive and use to build the control architecture.

5. CONCLUSIONS

An overview on behaviour-based robotics has been presented. Also, four of the most conceptually important behaviour-based architectures have been described with simulation examples. Designers of these systems must develop some interesting ideas and methods concerning how to design an AUV's control system. For future work, a complete control architecture is to be implemented and tested with real missions with GARBI and also with new small AUV named ICTINI. The control architecture, the Object Oriented Control Architecture for Autonomy, has been designed with a behavioural-based control layer that will give reactivity to the vehicle.

6. ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of the Spanish and British Governments for the "Accion Integrada" programme HB1997-0193. Also the CICYT for the financial project MAR97-0925C0202.

7. REFERENCES

Amat, J., J. Batlle, A. Casals and J. Forest (1996). GARBI: a low cost ROV, constrains and solutions. In: *6ème Seminaire IARP en robotique sous-marine*. pp. 1-22, Toulon-La Seyne, France.

Arkin, R. C. (1989). *Neuroscience in Motion: The application of Schema Theory to Mobile Robotics.* New York: Plenum Press.

Arkin, R. C. (1992). Behavior-Based Robot Navigation for Extended Domains. *Adaptative Behavior*, 1 (9),pp. 201-225.

Arkin, R. C. (1998). *Behavior-based Robotics.* Massachusetts Institute of Technology: MIT Press.

Balch, T. and R.C. Arkin (1993). Avoiding the Past: A simple but effective strategy for Reactive Navigation. *Proceedings of the IEEE International Conference on Robotics and Automation*. pp. 678-85, Atlanta, GA.

Boswell, A. J. and J.R. LEANEY (1994). Using the subsumption architecture in an autonomous underwater robot. In: *International Advanced Robotics program, Workshop on Mobile Robots for Subsea Environments*. Monterey, California, USA.

Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. In: *IEEE Journal of Robotics and Automation*, RA-2 (1),pp. 14-23.

Connell, J.H. (1989). Minimalist Mobile Robotics. Academic Press.

Coste-Maniere, E., H.H. Wang and A. Peuch (1995). Control architectures: what's going on? In: *Proceedings of the US-Portugal Workshop on Undersea Robotics and Intelligent Control*. pp. 54-60, Lisbon, Portugal.

Fossen, T. I. (1995). *Guidance and Control of Ocean vehicles.* John Wiley & Sons.

Goheen, K. (1995). Techniques for URV Modelling. *In: Underwater Robotics Vehicles* ed. J. Yuh. TSI Press.

Maes, P. (1989). How to do the right thing. *Connection Science Journal*, Vol. 1, No. 3, pp. 291-323.

Maes, P. (1990). Situated Agents Can Have Goals. In: *Robotics and Automation Systems*, 6 p. 49-70.

Pfeifer, R. and C. Scheier (1999). *Understanding Intelligence.* Massachusetts: MIT Press.

Ridao, P., J. Batlle, J. Amat and G.N. Roberts (1999). Recent trends in control architectures for autonomous underwater vehicles. In: *International Journal of Systems Science*, 30 (9),pp. 1033-1056.

Steels, L. (1992). The PDL reference manual. VUB AI Lab, Memo 92-5. Brussels, Belgium.

Valavanis, K. P., D. Gracanin, M. Matijasevic, R. Kolluru and G.A. Demetriou (1997). Control architectures for autonomous underwater vehicles. In: *IEEE Control Systems Magazine*, 17 (6),pp. 48-64.

Warren, C. W. (1990). A technique for autonomous underwater vehicle route planning. In: *IEEE Journal of Oceanic Engineering*, 15 (3), 199-204.